UNCLASSIFIED

AD 268 572

Reproduced
by the

ARMED SERVICES TECHNICAL INFORMATION AGENCY
ARLINGTON HALL STATION
ARLINGTON 12, VIRGINIA

UNCLASSIFIED

NOTICE: When government or other drawings, specifications or other data are used for any purpose other than in connection with a definitely related government procurement operation, the U. S. Government thereby incurs no responsibility, nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use or sell any patented invention that may in any way be related thereto.

62-1-5
XEROX

CATALOGED BY ASTIA
AS AD NO. 268572

D1-82-0141

BOEING SCIENTIFIC RESEARCH LABORATORIES

127 450

Procedures For Generating Normal Random Variables, II

G. Marsaglia

Mathematics Research

Also Available from Author

November 1961

ASTIA
DEC 28 1961
JIPDR

D1-82-0141

PROCEDURES FOR GENERATING NORMAL RANDOM VARIABLES, II

by

G. Marsaglia

Mathematical Note No. 243

Mathematics Research Laboratory

BOEING SCIENTIFIC RESEARCH LABORATORIES

November 1961

1. Introduction

A method for generating a normal random variable in terms of uniform random variables will be described. The method is based on representing a density function as a mixture of simpler densities, as outlined in [1]. It is quite fast and requires little storage (60 constants). It is not quite as fast as the method of [2], but it is simpler, with less chance for prospective users being set adrift in a sea of details.

We will fashion the procedure with use on a computer with numbers having 35 binary digits (bits) in mind. The details may be modified for numbers of different lengths or for decimal computers.

The normal random variable x is generated in terms of independent uniform random variables $u_1, u_2, \ldots$. Roughly, the procedure is this: we put $x = \frac{1}{2}u$ with frequency 35%; $x = \frac{1}{2} + \frac{1}{2}u$, 24%; $x = 1 + \frac{1}{2}u$, 13%; $x = 3/2 + \frac{1}{2}u$, 5%; and $x = 2 + \frac{1}{2}u$, with frequency about 1%. Thus with probability about .8 we put $x = a + \frac{1}{2}u$, the first 9 bits of u_1 being used to choose a from among 0, 1/2, 1, 3/2, 2 and the remaining 26 bits of u_1 used to form $\frac{1}{2}u$.

Of the remaining 20% we use the modified rejection technique given in [3] with frequency 19% and the other 1% is devoted to the tail. The modified rejection technique is applied to one of regions 6-10 above the rectangles in Fig. 1, while rectangles 1-5 provide the representations $x = 0 + \frac{1}{2}u$, $\frac{1}{2} + \frac{1}{2}u$, etc. We use the polar representation of a random normal point to dispose of the tail.

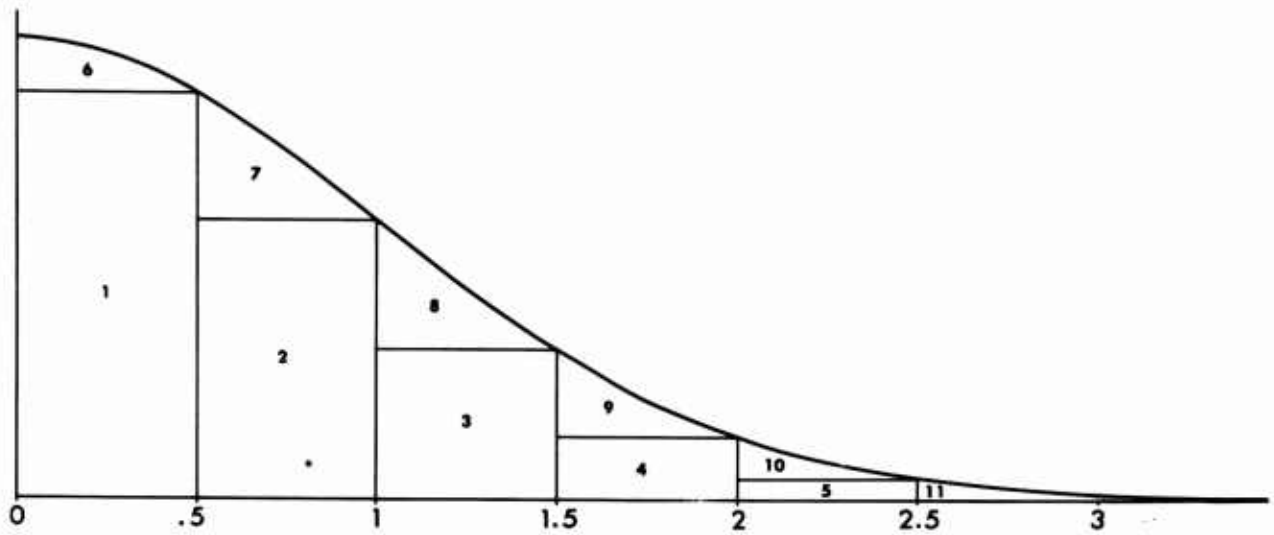


Fig. 1

The procedure has, in theory, unlimited accuracy; in practice, its accuracy is limited by the capabilities of the machine in single precision floating point.

2. The Rectangles

Let f be the absolute normal density,

$$f(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}, \quad x > 0.$$

We generate x with density f, attaching a random $\pm$ at some convenient point in the program. We write f as a mixture of 11 densities:

$$f(x) = \sum_{i=1}^{11} p_i g_i(x).$$

Fig. 1 shows how the terms in the mixture are stacked to form f.

The 5 rectangles provide the fast part of the program; $g_1,\ldots,g_5$ being rectangles with base $\frac{1}{2}$, an x with such a density may be found by putting $x = a + \frac{1}{2}u$. For example, region 1 of Fig. 1 provides the representation $x = 0 + \frac{1}{2}u$, and this should be used with frequency equal to the area of region 1, $p_1 = .352065326764299$. But it is wasteful to test $u_1 < p_1$ in its full form, as this will require all 35 bits of u_1; much better is to use only, say, 9 bits of u_1 to test $u_1 < \hat{p}_1$ where $\hat{p}_1$ is the nearest we can get to p_1 using 9 bits, $\hat{p}_1 = 180/512 = .3515625$, then we may use the last 26 bits of u_1 to form $\frac{1}{2}u$ and put $x = \frac{1}{2}u$. Later, with probability $p_1 - \hat{p}_1$, we will put $x = \frac{1}{2}u_2$, using a completely new uniform number u_2. We then will get by most of the time with a single uniform number u_1. Each of the probabilities p_1,p_2,p_3,p_4,p_5

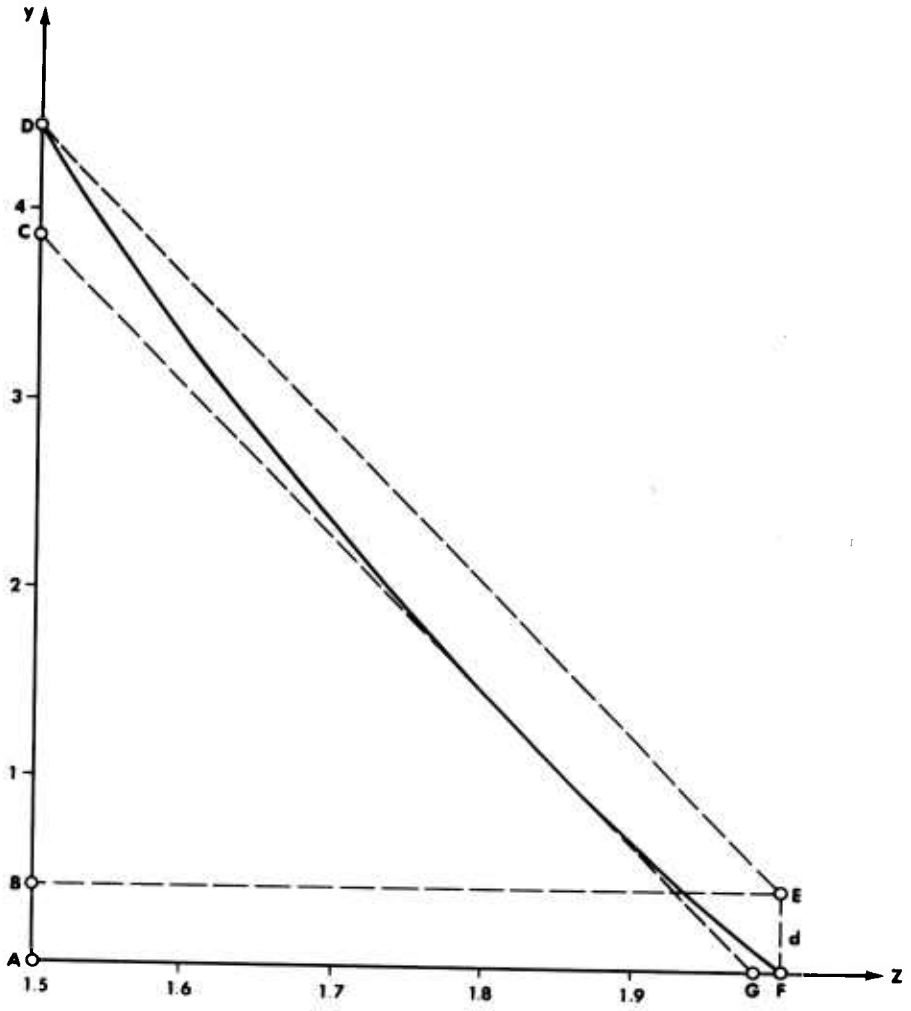


Fig. 2

is broken up in this way; the numerical values appear in the steps of the program and in the summary of constants.

3. The "Nearly Linear" Method

We illustrate the nearly linear method with region 9 of Fig. 1. It represents the term $p_9 g_9$ in the mixture, so we must, with probability $p_9 = .034123172128170$, generate a random variable with density g_9. This density is drawn in Fig. 2. Our method is to enclose the area under g in the trapezoid ADEF. The slope of CG and DE is -8. We choose a point (z,y) uniformly from the trapezoid and set $x = z$ if $y < g(z)$, but choose a new point (z,y) if not. We test to see if the point (z,y) is in triangle ACG ($y <$ const. $- 8z$), hoping to avoid testing $y < g(z)$. If that test is necessary, we take advantage of the exponential content of g, writing

$$g(z) = \frac{2}{p_9\sqrt{2\pi}}\left[e^{-\frac{1}{2}z^2} - e^{-2}\right] = \frac{2e^{-9/8}}{p_9\sqrt{2\pi}}\left[e^{-\frac{1}{2}(z^2-9/4)} - e^{-7/8}\right].$$

Then the test $y < g(z)$ hinges on whether this infinite sum is positive or negative:

$$ry - s + t - \frac{t^2}{2!} + \frac{t^3}{3!} - \frac{t^4}{4!} + \cdots$$

with $r = p_9\sqrt{\pi/2}\, e^{9/8}$, $s = 1 - e^{-7/8}$, $t = \frac{1}{2}(z + 3/2)(z - 3/2)$, $0 \le t \le 7/8$. This is an alternating series that converges rather quickly, and its sum is negative if and only if two successive partial sums are negative; positive if and only if two successive partial sums are positive. We should average only a few multiplications in order to

decide $y < g(z)$ with "unlimited" accuracy. Another advantage is that the structure of the test for g_9 is the same as that for the other g's, only r, s, and t changing.

The point (z,y) in the trapezoid of Fig. 2 may be produced quite fast, putting

$$z = \tfrac{1}{2}\min(u_2,u_3) + 1.5,$$

$$y = d + 4\max(u_2,u_3)$$

(choosing from triangle BDE) with probability $\frac{2}{2+d}$ and putting

$$z = \tfrac{1}{2}u_2,$$

$$y = du_3$$

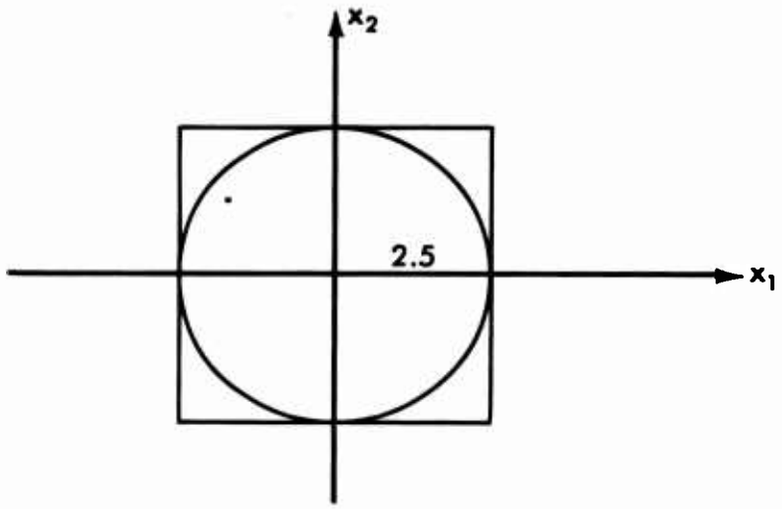
(choosing from ABEF) with probability $\frac{d}{2+d}$. If d is small we will avoid conventional multiplication most of the time, and by properly choosing d we will need only a few bits of u_1 to choose between the two methods for generating (z,y), so that the remaining bits of u_1 may be used to form u_2 and u_3.

For example, $p_9 = .0341231721281 70$. We let $\hat{p}_9 = 17/512 = .033203125$, then generate x with density g_9 with probability $\hat{p}_9$. In the course of the program, the test for this contingency is $\frac{460}{512} \le u_1 < \frac{460 + 17}{512}$. If u_1 is in this range, then by making $\frac{2}{2+d} = \frac{14}{17}$, we may test $u_1 < \frac{460 + 14}{512}$, keeping the last 26 bits of u_1 available for forming u_2 and u_3. This makes $d = 3/7 = .42857...$ whereas the smallest possible value for d is $d = .427...$. The loss is small, and the gain from not having to generate new u_2 and u_3 is great. We must generate x

from density g_9 with probability $p_9 - \hat{p}_9$ later on, using new u_2 and u_3, but the average time loss for this compensating step is negligible.

4. The Tail

We need x conditioned by $|x| > 2.5$. Consider the plane of points (x_1,x_2) with x_1 and x_2 independent standard normal.



We want a point (x_1,x_2) outside the square, then one of the coordinates will serve as x. We use the polar representation (ρ,θ) to choose points outside the circle until we get one outside the square, then change to rectangular coordinates (x_1,x_2).

The exterior of the circle has measure .044, the exterior of the square, .025. The ratio of the two is about .57 so that 57% of the time a point lying outside the circle will lie outside the square. If (x_1,x_2) lies outside the square, only once in 150 times will both $|x_1|$ and $|x_2|$ be > 2.5, so it probably isn't worth the trouble to store the extra one

for future use, as this would mean that each time the tail was encountered in the program, the request for a possible stored value of x would almost always be denied.

It is elementary to verify that putting

$$x_1 = v_1 \left(\frac{25/4 + 2w}{v_1^2 + v_2^2} \right)^{\frac{1}{2}},$$

$$x_2 = v_2 \left(\frac{25/4 + 2w}{v_1^2 + v_2^2} \right)^{\frac{1}{2}},$$

with w exponential and v_1, v_2 uniform on $(-1,1)$ and conditioned by $v_1^2 + v_2^2 < 1$, will provide a point (x_1, x_2) outside the circle with the appropriate distribution.

5. Outline of a Program

We give a series of steps outlining a program based on the above procedure, assuming we have a subroutine which, when called for, will generate a new uniform number u in terms of the current u; for example, by putting: new $u = 2^k u + u + 1 \pmod{2^{35}}$, as suggested by Rotenberg [5].

1. If $0 \le u_1 < \frac{180}{512}$ put $x = \frac{1}{2}u$, with $\frac{1}{2}u$ formed from bits 10-35 of u_1.

2. If $\frac{180}{512} \le u_1 < \frac{303}{512}$ put $x = \frac{1}{2} + \frac{1}{2}u$, with $\frac{1}{2}u$ formed from bits 10-35 of u_1.

3. If $\frac{303}{512} \le u_1 < \frac{369}{512}$ put $x = 1 + \frac{1}{2}u$, with $\frac{1}{2}u$ formed from bits 10-35 of u_1.

4. If $\frac{369}{512} \le u_1 < \frac{396}{512}$ put $x = 1.5 + \frac{1}{2}u$, with $\frac{1}{2}u$ formed from bits 10-35 of u_1.

5. If $\frac{396}{512} \le u_1 < \frac{404}{512}$ put $x = 2.0 + \frac{1}{2}u$, with $\frac{1}{2}u$ formed from bits 10-35 of u_1.

6. If $\frac{404}{512} \le u_1 < \frac{433}{512}$ put $j = 7$, if $\frac{433}{512} \le u_1 < \frac{460}{512}$ put $j = 8$, if $\frac{460}{512} \le u_1 < \frac{477}{512}$ put $j = 9$, if $\frac{477}{512} \le u_1 < \frac{492}{512}$ put $j = 6$, and if $\frac{492}{512} \le u_1 < \frac{499}{512}$ put $j = 10$. Test $u_1 < q_j$, if yes, form $z = a_j + \frac{1}{2}\min(u_2,u_3)$ and $y = d_j + 4\max(u_2,u_3)$; if no, form $z = a_j + \frac{1}{2}u_2$, $y = d_j u_3$. In either case, u_2 is formed from bits 10-22 and u_3 from bits 23-35 of u_1. [K] If $y < e_j - 8z$, put $x = z$; if not, let $t = \frac{1}{2}(z + a_j)(z - a_j)$ and form terms of the sequence

$$r_j y - s_j,\ r_j y - s_j + t,\ r_j y - s_j + t - \frac{t^2}{2!},\ r_j y - s_j + t - \frac{t^2}{2!} + \frac{t^3}{3!}, \ldots$$

until two successive terms have the same sign. If two successive terms are negative, put $x = z$, if two successive terms are positive, go to H.

7. If $\frac{499}{512} \le u_1 < .980619788956091$ choose i from $1,2,\ldots,5$ with probability $p_i - \hat{p}_i$, generate a uniform number u_2 and put $x = \frac{1}{2}(i-1) + \frac{1}{2}u_2$.

8. If $.980619788956091 \leq u_1 < .987580669348448$ choose j from among 6,7,8,9,10 with probability $p_j - \hat{p}_j$ and go to H.

9. If $.987580669348448 \leq u_1 < 1$, form

$$x_1 = v_1\left(\frac{25/4 + 2w}{v_1^2 + v_2^2}\right)^{\frac{1}{2}},$$

$$x_2 = v_2\left(\frac{25/4 + 2w}{v_1^2 + v_2^2}\right)^{\frac{1}{2}},$$

where w has the exponential distribution and v_1, v_2 are uniform on $(-1,1)$, conditioned by $v_1^2 + v_2^2 < 1$. Test $|x_1| > 2.5$; if yes, put $x = x_1$. If no, test $|x_2| > 2.5$; if yes, put $x = x_2$; if no, generate a new pair x_1, x_2 and try again.

H. Generate a new uniform number u. Test $u < b_j$, if yes, let $z = a_j + \frac{1}{2}\min(u_2,u_3)$, $y = d_j + 4\max(u_2,u_3)$; if no, let $z = a_j + \frac{1}{2}u_2$, $y = d_j u_3$ where u_2 and u_3 are formed from the first and last parts of a new uniform number generated from u. Now go to [K] of step 6.

6. Remarks

With probability $\cdot\frac{404}{512} = .79$ we will use the representation $x = a + \frac{1}{2}u$ of steps 1-5, and this is the principal reason for a fast program. However, even when, with frequency $\frac{95}{512}$ or about 19%, we require step 6, the time loss is not crushing, for most of the time we will get through step 6 without using a single conventional multiplication and will require only one uniform number u_1.

It is in trying to avoid the use of more than one uniform number in step 6 that we make our only compromise with accuracy, for we use only 13 bits to represent numbers on an interval of length $\frac{1}{2}$. A program based on the outline above can be described as being as accurate as the single precision capacity of the machine, except that with probability $\frac{1}{5}$ the value of x may differ by about 2^{-14} from the value that should be delivered. Any nuts wanting accuracy greater than this can call up a new uniform number at that point, with a loss of perhaps 15 cycles, or a net loss of about $\frac{15}{5} = 3$ cycles in the average running time of the program.

Step 7 corrects for our inability to handle regions 1 to 5 using only 9 bits, and step 8 does the same for regions 6 to 10. Step 9 provides an x from the tail. Note that in step 9 a signed value of x is returned, whereas in the other steps a positive x is returned and a random $\pm$ must be assigned at some convenient point. The method for doing this is not critical in steps 7 or 8, but in the first 6 steps, particularly steps 1-5, the method should be chosen so that time is not wasted in this assignment.

Step 9 requires an exponential variable w. The ease with which w can be produced is probably more important than saving time here, since step 9 provides only one x in a hundred. If a logarithm subroutine is available, $w = -\ln u$ will serve, or the method in [4] may be used.

SUMMARY OF CONSTANTS

j	b_j	d_j	e_j	q_j	$\hat{p}_j$	a_j
6	12/15	1/2	3.0380	489/512	15/512=.029296875	0
7	28/29	1/14	7.8099	432/512	29/512=.056640625	1/2
8	25/27	4/25	11.930	458/512	27/512=.052734375	1
9	14/17	3/7	15.850	474/512	17/512=.033203125	3/2
10	5/7	4/5	19.780	497/512	7/512=.013671875	2

j	p_j	$p_j-\hat{p}_j$	r_j	s_j
6	.030859595783727	.001562720783727	.038676767667587	.117503097415405
7	.057793845069917	.001153220069917	.082078297231197	.312710721209028
8	.054178509659306	.001444134659306	.111952612794322	.464738571481000
9	.034123172128170	.000920047128170	.131731800427316	.583137980321492
10	.015552632751237	.001880757751237	.144029953116673	.675347532641650

i	p_i	$\hat{p}_i$	$p_i-\hat{p}_i$
1	.352065326764299	180/512=.3515625	.000502826764299
2	.241970724519143	123/512=.240234375	.001736349519143
3	.129517595665892	66/512=.12890625	.000611345665892
4	.053990966513188	27/512=.052734375	.001256591513188
5	.017528300493569	8/512=.015625	.001903300493569

REFERENCES

[1] G. Marsaglia, "Expressing a Random Variable in Terms of Uniform Random Variables", Annals of Mathematical Statistics, vol. 32, no. 3, Sept. 1961, pp. 894-898.

[2] G. Marsaglia and M. D. MacLaren, "A Program for Generating Normal Random Variables", to appear.

[3] G. Marsaglia, "Remark on Generating a Random Variable Having a Nearly Linear Density Function", Boeing Scientific Research Laboratories Document D1-82-0107, May 1961.

[4] G. Marsaglia, "Generating Exponential Random Variables", Annals of Mathematical Statistics, vol. 32, no. 3, Sept. 1961, pp. 899-900.

[5] A. Rotenberg, "A New Pseudo-Random Generator", Journal of the Association for Computing Machinery, vol. 7, no. 1, Jan. 1960, pp. 75-77.